# DSP report

Kanae Kurita, Ankit Anand, Omkar Padwal

January 2022

## 1 Abstract

The report shows the use of IIR filter to remove noise from real-time data, and use that data for detection. We have successfully created the programs that make it possible.

## 2 Data Acquisition

For this assignment we have used webcam to collect data in real-time. Using the webcam2rgb program real-time data is recorded, which was then split into red, green, and blue signals. These signals were then used for the purpose of the application given below in the report.

## 3 Real-time filtering

### 3.1 Application

In this report, we created a program to detect ambulances, from sirens and blinks of blue and red light. If ambulance detection is possible, it may help to reach its destination as soon as possible. One example is transportation network optimization like adjusting the timing of traffic lights by the detection. Alternatively, if a general vehicle detects it, it may be possible to efficiently give way to an ambulance or prevent an unexpected traffic accident. We made it possible by using webcam and RGB detection.

### 3.2 Use of IIR filter for the application

The raw data recorded with the help of webcam consist of noise. This data is filtered using the IIR filter. The output then can be used for detection. For the purpose of filtering we have used 2nd order IIR filter and chain of 2nd order IIR filter.

#### 3.2.1 Second Order IIR filter

A second order IIR filter is a filter which has two time delays. The equation of second order is given as below:

$$H(z) = \frac{b[0] + b[1]z^{-1} + b[2]z^{-2}}{1 + a[1]z^{-1} + a[2]z^{-2}} \tag{1}$$

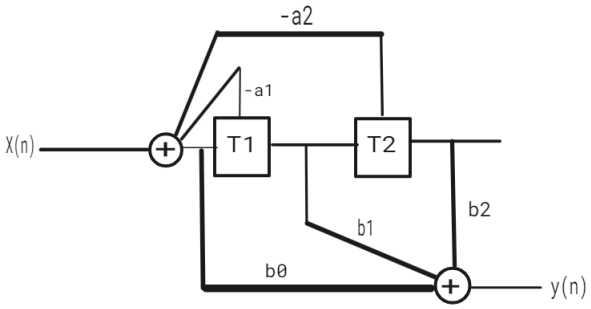


Figure 1

#### 3.2.2 Chain of Second Order Filter

Chain of second order filter consists of two second order filters connected in series. It is shown below:

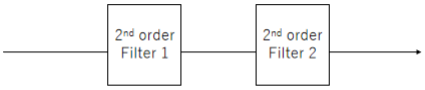


Figure 2

### 3.3 Implementation of the application

The webcam records the video and separates the red, blue, and green elements. The elements are then passed into the IIR filters to get the filtered output. The detection works in the following way:

1. Intensity of red is higher when red light is blinking
2. Intensity of blue is higher when blue light is blinking

In case of an ambulance both the lights blink periodically, which means that the intensity of blue and red light will increase and decrease periodically. If this kind of pattern is found then the presence of an ambulance can be detected.

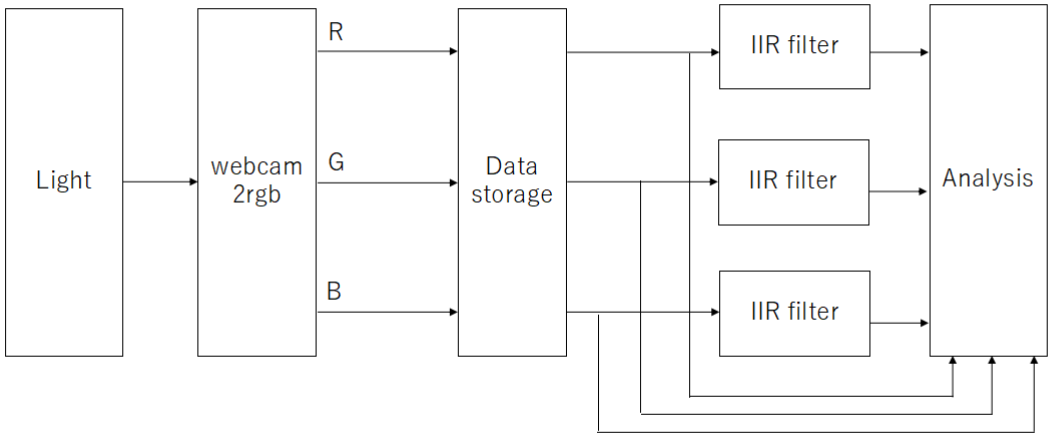


Figure 3: Dataflow diagram of application

### 3.4 Calculation of Impulse response

The equation of impulse response for a second order filter is given above. In the equation there are five coefficients. Out of the five coefficients the three coefficients in the numerator part are of FIR filter and the denominator part is of IIR filter. SOS coefficients for a bandpass filter were generated using Python's high level command "buffer". When we visualized the frequency component of the signal, it was observed that the frequency contained an unnecessary component in 1 to 4Hz. Each coefficient of formula (1) is as follows:

b[0] = 0.07,
b[1] = 0.13,
b[2] = 0.07,
a[1] = -1.11,
a[2] = 0.52,

### 3.5 Original Signal Plots

The plots of red and blue lights of an ambulance's siren are shown below

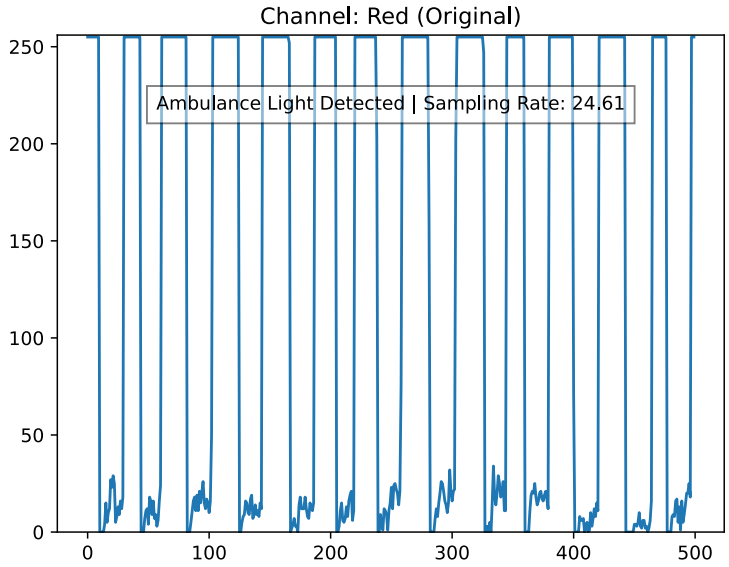


Figure 4: Red Colour Signal

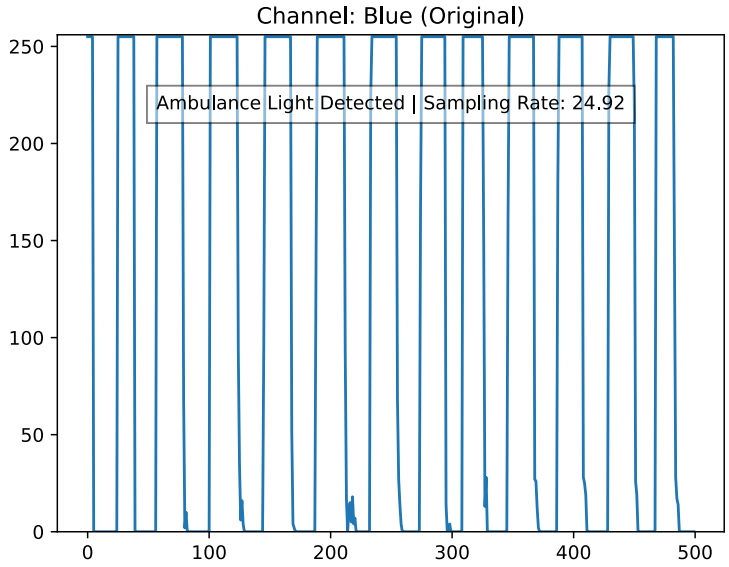


Figure 5: Blue Colour Signal

### 3.6 Filtered Signal Plots

By taking reference of figure 1 we calculated the filter output from the original output. There are two summations, one is input summation and the other one is output summation. The input summation is adding the input, a1*time_delay, and a2*time_delay. The output summation is adding three coefficients: b0, b1, and b2. The equation for input and output can be written as below:

$$input = x - a1 * z^{-1} - a2 * z^{-2} \tag{2}$$

$$output = input * b0 + b1 * z^{-1} + b2 * z^{-2} \tag{3}$$

Using this, IIR filter was created. The function of IIR filter built in python, which was used in the program is given below:

```
def filter(self,v):
    """Sample by sample filtering
    v -- scalar sample
    returns filtered sample
    """
    input = v - (self.denominator1 * self.buffer1) - (self.denominator2 * self.buffer2)
    output = (self.numerator1 * self.buffer1) + (self.numerator2 * self.buffer2) + input * self.numer
```

```
        self.buffer2 = self.buffer1
        self.buffer1 = input
        return output
```

For the process of filtering a SOS band-pass filter was used. The cut-off frequencies used for this purpose were 1Hz and 4Hz. The SOS coefficients were calculated using the 'butter' function and then they were used to calculate the impulse response. Using the impulse response we got a filtered output.

```
sos2 = sos1 = signal.butter(2, [1, 4], 'bandpass', fs=30, output='sos') #band pass
```

The plots of red and blue lights after passed through a second order bandpass filter are shown below.

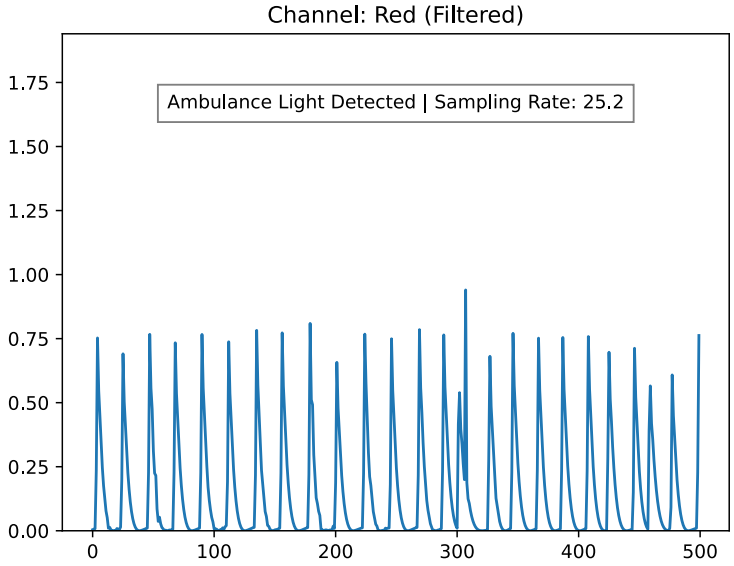


Figure 6: Filtered red output

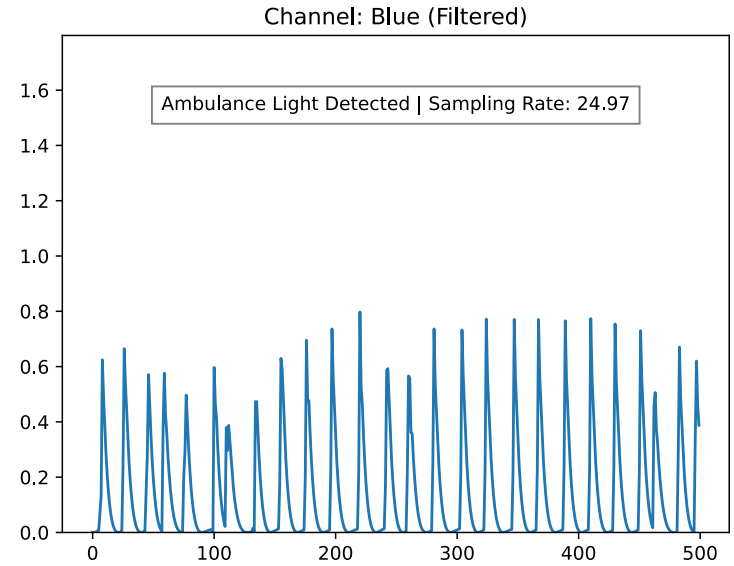


Figure 7: Filtered Blue output

## 3.7 Process of Detection

The detection was done by using a threshold intensity for both blue and red. The threshold used in the code is 0.2. If the intensity of blue and red is above 0.2 then ambulance is present, if not then it is not there. Above task was obtained using the following logic:
Function used to detect the threshold:

```
def detect_ambulance(rf, bf, gf):
```

```
if rf > 0.2 or bf > 0.2:
    return "Ambulance Light Detected"
else:
    return "Ambulance Light Not Detected"
```

The values rf, bf, and gf were calculated using the code stated below:

```
dataF = iir2.filter(data) #Band pass Filtering the input signal
dataFR = iir3.filter(dataF) #Adding wavelet to detect peaks
dataFR = dataFR * dataFR #squaring to remove noise
bf = dataFR[0]
gf = dataFR[1]
rf = dataFR[2]
```

Below plot shows an example in which yellow light was used instead of red and blue light.

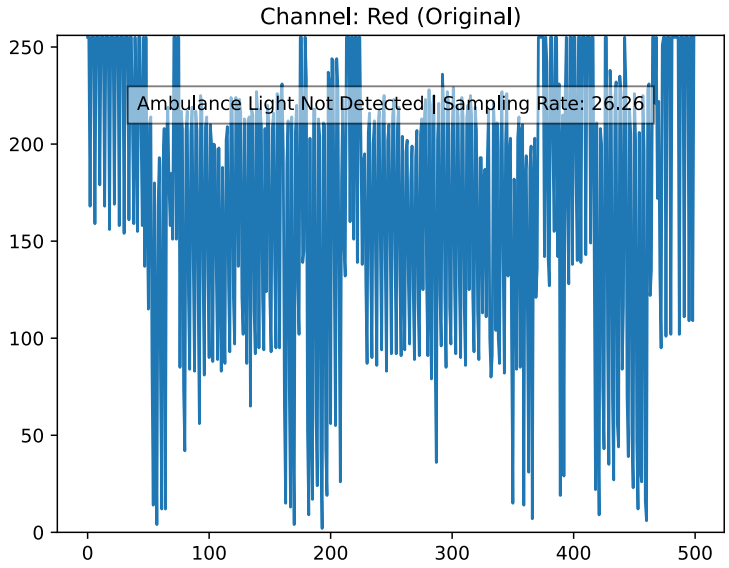


Figure 8: Red colour plot for yellow light

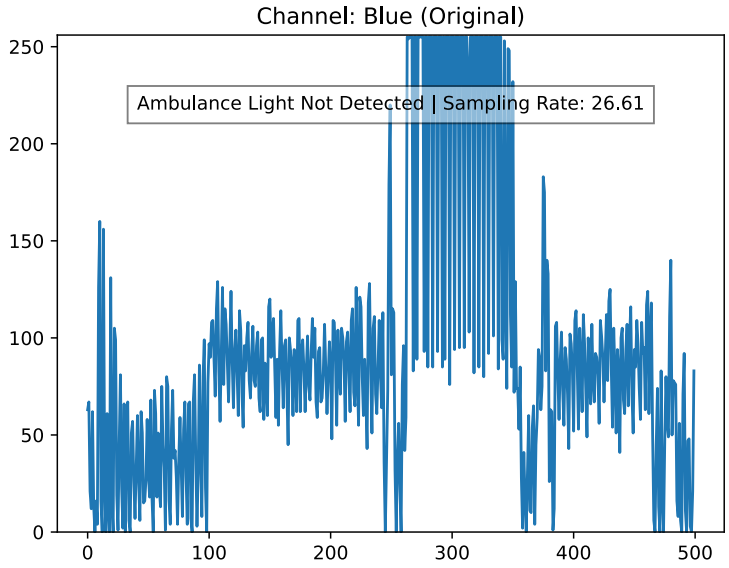


Figure 9: Blue colour plot for yellow light

After filtering the original signal shown above, we got the following results:

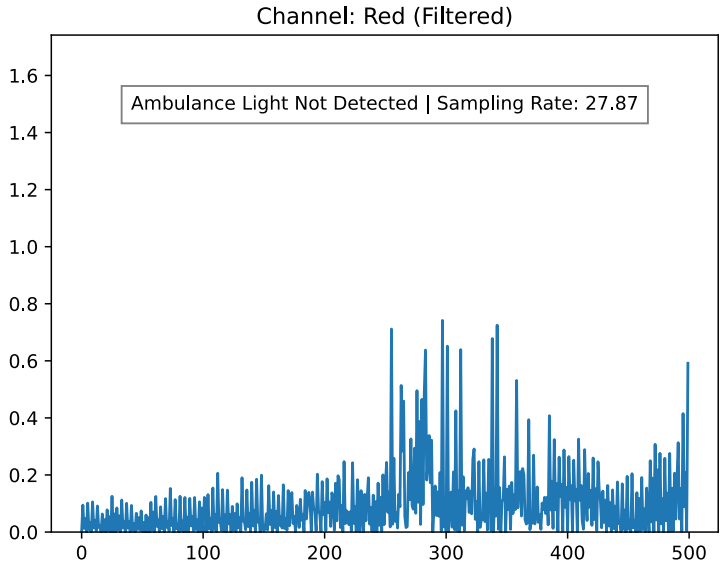


Figure 10: Filtered Red Colour plot for yellow light

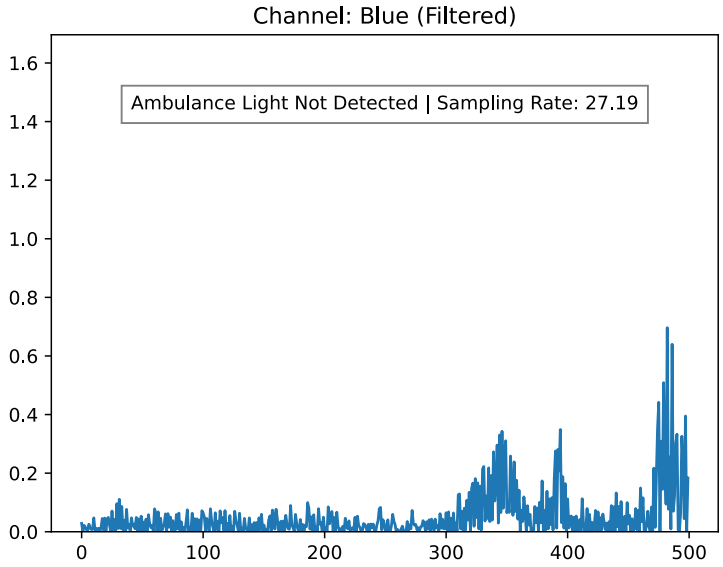


Figure 11: Filtered Blue Colour plot for yellow light

When yellow light was used instead of red and blue, ambulance was not detected. This was because the intensity of of filtered red and blue colour for yellow colour was less than 0.2.

# 4 Appendix

## 4.1 Python Code for Realtime Filtering

```
    #!/usr/bin/env python3
# -*- coding: utf-8 -*-

import webcam2rgb
import numpy as np

import matplotlib.pyplot as plt
import matplotlib.animation as animation

from scipy import signal
from scipy.fft import fft, ifft

import iir_filter
import time

class RealtimePlotWindow:

    def __init__(self, channel: str):
        # create a plot window
        self.fig, self.ax = plt.subplots()
        plt.title(f"Channel: {channel}")
        self.title = self.ax.text(0.5,0.85, "", bbox={'facecolor':'w', 'alpha':0.5, 'pad':5},
                transform=self.ax.transAxes, ha="center")
        # that's our plotbuffer
        self.plotbuffer = np.zeros(500)
        # create an empty line
        self.line, = self.ax.plot(self.plotbuffer)
        # axis
        self.ax.set_ylim(0, 1)
        # That's our ringbuffer which accumluates the samples
        # It's emptied every time when the plot window below
        # does a repaint
        self.ringbuffer = []
        # add any initialisation code here (filters etc)
        # start the animation
        self.ani = animation.FuncAnimation(self.fig, self.update, interval=100)


    # updates the plot
    def update(self, data):
        # add new data to the buffer
        self.plotbuffer = np.append(self.plotbuffer, self.ringbuffer)
        # only keep the 500 newest ones and discard the old ones
        self.plotbuffer = self.plotbuffer[-500:]
        self.ringbuffer = []
        # set the new 500 points of channel 9
        self.line.set_ydata(self.plotbuffer)
        self.ax.set_ylim(0, max(self.plotbuffer)+1)
        return self.line,

    # appends data to the ringbuffer
    def addData(self, v, sr, flag):
        self.ringbuffer.append(v)
        self.title.set_text(f"{flag} | Sampling Rate: {round(sr, 2)}")

realtimePlotWindowRedOg = RealtimePlotWindow("Red (Original)")
realtimePlotWindowBlueOg = RealtimePlotWindow("Blue (Original)")
realtimePlotWindowGreenOg = RealtimePlotWindow("Green (Original)")
realtimePlotWindowBlue = RealtimePlotWindow("Blue (Filtered)")
```

```
realtimePlotWindowGreen = RealtimePlotWindow("Green (Filtered)")
realtimePlotWindowRed = RealtimePlotWindow("Red (Filtered)")

redd = []
greenn = []
bluee = []
reddf = []
greennf = []
blueef = []
#sos1 = signal.butter(2, [0.4, 1], 'bandpass', fs=30, output='sos') #band pass
sos2 = sos1 = signal.butter(2, [1, 4], 'bandpass', fs=30, output='sos') #band pass

#iir1 = iir_filter.IIR2_filter(sos1[0])
iir2 = iir_filter.IIR2_filter(sos2[0])

start_time = time.time()

def create_wavelet(N, fs):
    duration = N / fs
    t = np.linspace(-duration/2, duration/2, N)
    wavelet = np.sinc(t*fs)
    return wavelet

wavelet = create_wavelet(30, 30)
iir3 = iir_filter.IIR2_filter(wavelet)

def detect_ambulance(rf, bf, gf):
    if rf > 0.2 or bf > 0.2:
        return "Ambulance Light Detected"
    else:
        return "Ambulance Light Not Detected"

#create callback method reading camera and plotting in windows
def hasData(retval, data):
    b = data[0]
    g = data[1]
    r = data[2]


    dataF = iir2.filter(data) #Band pass Filtering the input signal
    dataFR = iir3.filter(dataF) #Adding wavelet to detect peaks
    dataFR = dataFR * dataFR #squaring to remove noise
    bf = dataFR[0]
    gf = dataFR[1]
    rf = dataFR[2]

    flag = detect_ambulance(rf, bf, gf)
    redd.append(r)

    elapsed_time = time.time() - start_time #tracking elapsed time to check sampling rate
    sr = len(redd)/elapsed_time #ACTUAL SAMPLING RATE OF THE CAMERA
    realtimePlotWindowRedOg.addData(r, sr, flag)
    realtimePlotWindowBlueOg.addData(b, sr, flag)
    realtimePlotWindowGreenOg.addData(g, sr, flag)
    realtimePlotWindowBlue.addData(bf, sr, flag)
    realtimePlotWindowGreen.addData(gf, sr, flag)
    realtimePlotWindowRed.addData(rf, sr, flag)


#create instances of camera
camera = webcam2rgb.Webcam2rgb()
#start the thread and stop it when we close the plot windows
camera.start(callback = hasData, cameraNumber=0, directShow = True)
```

```
print("camera samplerate: ", camera.cameraFs(), "Hz")
plt.show()
camera.stop()

print('finished')
```

### 4.2 Python code IIR filter class

```
    import numpy as np
from scipy import signal
import numpy as np

class IIR2_filter:
    """2nd order IIR filter"""

    def __init__(self,s):
        """Instantiates a 2nd order IIR filter
        s -- numerator and denominator coefficients
        """
        self.numerator0 = s[0]
        self.numerator1 = s[1]
        self.numerator2 = s[2]
        self.denominator1 = s[4]
        self.denominator2 = s[5]
        self.buffer1 = 0
        self.buffer2 = 0

    def filter(self,v):
        """Sample by sample filtering
        v -- scalar sample
        returns filtered sample
        """
        input = v - (self.denominator1 * self.buffer1) - (self.denominator2 * self.buffer2)
        output = (self.numerator1 * self.buffer1) + (self.numerator2 * self.buffer2) + input * self.numer
        self.buffer2 = self.buffer1
        self.buffer1 = input
        return output

    def lowPassTest(self, s, input):
        x_sos_filt = np.round(signal.sosfilt(s, input), 2)
        x_iir2_filt = []
        for i in input:
            x_iir2_filt.append(np.round(self.filter(i), 2))

        print("Low Pass Filter O/P SOSFilt= ", x_sos_filt)
        print("Low Pass Filter O/P IIR2 = ", x_iir2_filt)
        try:
            if np.testing.assert_equal(x_iir2_filt, x_sos_filt, err_msg='Not matching', verbose=True) ==
                return "Test Successfull. IIR filter performing as expected."
        except AssertionError:
            return "Test Unsuccessfull. IIR filter not performing as expected."

    def bandPassTest(self, s, input):
        x_sos_filt = np.round(signal.sosfilt(s, input), 2)
        x_iir2_filt = []
        for i in input:
            x_iir2_filt.append(np.round(self.filter(i), 2))

        print("Band Pass Filter O/P SOSFilt= ", x_sos_filt)
        print("Band Pass Filter O/P IIR2 = ", x_iir2_filt)
```

```
        try:
            if np.testing.assert_equal(x_iir2_filt, x_sos_filt, err_msg='Not matching', verbose=True) ==
                return "Test Successfull. IIR filter performing as expected."
        except AssertionError:
            return "Test Unsuccessfull. IIR filter not performing as expected."

    def bandStopTest(self, s, input):
        x_sos_filt = np.round(signal.sosfilt(s, input), 2)
        x_iir2_filt = []
        for i in input:
            x_iir2_filt.append(np.round(self.filter(i), 2))

        print("Band Stop Filter O/P SOSFilt= ", x_sos_filt)
        print("Band Stop Filter O/P IIR2 = ", x_iir2_filt)
        try:
            if np.testing.assert_equal(x_iir2_filt, x_sos_filt, err_msg='Not matching', verbose=True) ==
                return "Test Successfull. IIR filter performing as expected."
        except AssertionError:
            return "Test Unsuccessfull. IIR filter not performing as expected."

    def highPassTest(self, s, input):
        x_sos_filt = np.round(signal.sosfilt(s, input), 2)
        x_iir2_filt = []
        for i in input:
            x_iir2_filt.append(np.round(self.filter(i), 2))

        print("High Pass Filter O/P SOSFilt= ", x_sos_filt)
        print("High Pass Filter O/P IIR2 = ", x_iir2_filt)
        try:
            if np.testing.assert_equal(x_iir2_filt, x_sos_filt, err_msg='Not matching', verbose=True) ==
                return "Test Successfull. IIR filter performing as expected."
        except AssertionError:
            return "Test Unsuccessfull. IIR filter not performing as expected."

    def chainTestIIR(self, input):
        x_iir_filt = []
        for i in input:
            x_iir_filt.append(np.round(self.filter(i), 2))
        return x_iir_filt


class IIR_filter:
    """IIR filter"""
    def __init__(self,sos):
        """Instantiates an IIR filter of any order
        sos -- array of 2nd order IIR filter coefficients
        """
        self.cascade = []
        for s in sos:
            self.cascade.append(IIR2_filter(s))

    def filter(self,v):
        """Sample by sample filtering
        v -- scalar sample
        returns filtered sample
        """
        for f in self.cascade:
            v = f.filter(v)
        return v

    def lowPassTest(self, s, input):
        x_sos_filt = np.round(signal.sosfilt(s, input), 2)
        x_iir_filt = []
```

```
        for i in input:
            x_iir_filt.append(np.round(self.filter(i), 2))

        print("Low Pass Filter O/P SOSFilt= ", x_sos_filt)
        print("Low Pass Filter O/P IIR = ", x_iir_filt)
        try:
            if np.testing.assert_equal(x_iir_filt, x_sos_filt, err_msg='Not matching', verbose=True) == N
                return "Test Successfull. IIR filter performing as expected."
        except AssertionError:
            return "Test Unsuccessfull. IIR filter not performing as expected."

    def bandPassTest(self, s, input):
        x_sos_filt = np.round(signal.sosfilt(s, input), 2)
        x_iir_filt = []
        for i in input:
            x_iir_filt.append(np.round(self.filter(i), 2))

        print("Band Pass Filter O/P SOSFilt= ", x_sos_filt)
        print("Band Pass Filter O/P IIR = ", x_iir_filt)
        try:
            if np.testing.assert_equal(x_iir_filt, x_sos_filt, err_msg='Not matching', verbose=True) == N
                return "Test Successfull. IIR filter performing as expected."
        except AssertionError:
            return "Test Unsuccessfull. IIR filter not performing as expected."

    def bandStopTest(self, s, input):
        x_sos_filt = np.round(signal.sosfilt(s, input), 2)
        x_iir_filt = []
        for i in input:
            x_iir_filt.append(np.round(self.filter(i), 2))

        print("Band Stop Filter O/P SOSFilt= ", x_sos_filt)
        print("Band Stop Filter O/P IIR = ", x_iir_filt)
        try:
            if np.testing.assert_equal(x_iir_filt, x_sos_filt, err_msg='Not matching', verbose=True) == N
                return "Test Successfull. IIR filter performing as expected."
        except AssertionError:
            return "Test Unsuccessfull. IIR filter not performing as expected."

    def highPassTest(self, s, input):
        x_sos_filt = np.round(signal.sosfilt(s, input), 2)
        x_iir_filt = []
        for i in input:
            x_iir_filt.append(np.round(self.filter(i), 2))

        print("High Pass Filter O/P SOSFilt= ", x_sos_filt)
        print("High Pass Filter O/P IIR = ", x_iir_filt)
        try:
            if np.testing.assert_equal(x_iir_filt, x_sos_filt, err_msg='Not matching', verbose=True) == N
                return "Test Successfull. IIR filter performing as expected."
        except AssertionError:
            return "Test Unsuccessfull. IIR filter not performing as expected."

    def chainTestIIR(self, input):
        x_iir_filt = []
        for i in input:
            x_iir_filt.append(np.round(self.filter(i), 2))
        return x_iir_filt
```

### 4.3 Python code for Unit-test

```
import iir_filter
```

```
from scipy import signal
import numpy as np

class IIRTest:
    fs = 250
    fc = 10
    input = [-1.0, 0.5, 1.0]

    def lowPassTest(self):
        print("Commencing low pass unit test for the filters!")
        sos = signal.butter(1, self.fc, fs = self.fs, output='sos')
        iir1 = iir_filter.IIR_filter(sos)
        iir2 = iir_filter.IIR2_filter(sos[0])
        print(iir1.lowPassTest(sos, self.input))
        print(iir2.lowPassTest(sos, self.input))
        print("low pass test ended!")

    def bandPassTest(self):
        print("Commencing band pass unit test for the filters!")
        sos = signal.butter(1, [6, 8], fs = self.fs, btype='bandpass', output='sos')
        iir1 = iir_filter.IIR_filter(sos)
        iir2 = iir_filter.IIR2_filter(sos[0])
        print(iir1.bandPassTest(sos, self.input))
        print(iir2.bandPassTest(sos, self.input))

    def bandStopTest(self):
        print("Commencing band stop unit test for the filters!")
        sos = signal.butter(1, [6, 8], fs = self.fs, btype='bandstop', output='sos')
        iir1 = iir_filter.IIR_filter(sos)
        iir2 = iir_filter.IIR2_filter(sos[0])
        print(iir1.bandStopTest(sos, self.input))
        print(iir2.bandStopTest(sos, self.input))

    def highPassTest(self):
        print("Commencing high pass unit test for the filters!")
        sos = signal.butter(1, self.fc, fs = self.fs, btype='highpass', output='sos')
        iir1 = iir_filter.IIR_filter(sos)
        iir2 = iir_filter.IIR2_filter(sos[0])
        print(iir1.highPassTest(sos, self.input))
        print(iir2.highPassTest(sos, self.input))

    def chainTestIIR(self):
        sos1 = signal.butter(1, self.fc, fs = self.fs, output='sos')
        sos2 = signal.butter(1, 4, fs = self.fs, btype='highpass', output='sos')
        sos3 = signal.butter(1, [6, 8], fs = self.fs, btype = 'bandstop', output='sos')

        x_sos_filt = np.round(signal.sosfilt(sos3, signal.sosfilt(sos2, signal.sosfilt(sos1, self.input))

        iir1_1 = iir_filter.IIR_filter(sos1)
        iir1_2 = iir_filter.IIR_filter(sos2)
        iir1_3 = iir_filter.IIR_filter(sos3)

        iir2_1 = iir_filter.IIR2_filter(sos1[0])
        iir2_2 = iir_filter.IIR2_filter(sos2[0])
        iir2_3 = iir_filter.IIR2_filter(sos3[0])

        x_iir1_filt = iir1_3.chainTestIIR(iir1_2.chainTestIIR(iir1_1.chainTestIIR(self.input)))
        x_iir2_filt = iir2_3.chainTestIIR(iir2_2.chainTestIIR(iir2_1.chainTestIIR(self.input)))

        print("Chain Butterworth Filter O/P SOSFilt= ", x_sos_filt)
        print("Chain IIR Filter O/P = ", x_iir1_filt)
        print("Chain 2nd order IIR Filter O/P = ", x_iir2_filt)
        try:
```

```
            if np.testing.assert_equal(x_iir1_filt, x_sos_filt, err_msg='Not matching', verbose=True) ==
                print("Test Successfull. IIR filter chain performing as expected.")
        except AssertionError:
                print("Test Unsuccessfull. IIR filter chain not performing as expected.")
        try:
            if np.testing.assert_equal(x_iir1_filt, x_sos_filt, err_msg='Not matching', verbose=True) ==
                print("Test Successfull. IIR filter chain performing as expected.")
        except AssertionError:
                print("Test Unsuccessfull. IIR filter chain not performing as expected.")



iirTest = IIRTest()

iirTest.lowPassTest()
iirTest.bandPassTest()
iirTest.bandStopTest()
iirTest.highPassTest()
iirTest.chainTestIIR()
```